On the Computation and Verification Query Complexity of Symmetric Functions

Jayadev Acharya
EECS, MIT
jayadev@csail.mit.edu

Hirakendu Das
Yahoo
hdas@yahoo-inc.com

Ashkan Jafarpour
ECE, UCSD
ashkan@ucsd.edu

Alon Orlitksy
ECE & CSE, UCSD
alon@ucsd.edu

Ananda Theertha Suresh
ECE, UCSD
asuresh@ucsd.edu

Abstract

In the query model of multivariate function computation, the values of the inputs are queried sequentially in an order that may depend on previously revealed values until the function's value can be determined. The function's computation query complexity is the lowest expected number of queries required by any query order. Instead of computation, it is often easier to consider verification, where the value of the function is given and the queries aim to verify it. The lowest expected number of queries necessary is the function's verification query complexity. We show that for all symmetric functions of independent binary random variables, the computation and verification complexities coincide. This provides a simple method for finding the query complexity and optimal query order for computing many functions. We also show that after relaxing any of the symmetry, independence, or binary inputs restrictions, there are functions whose verification complexity is strictly lower than their computation complexity.

1 Introduction

Evaluating a multivariate function is a crucial and ubiquitous task whose importance has inspired countless different computational models. These models are made by restricting the function's class, changing the computational elements, moderating the computation error probability, adding assumptions on input probabilities, assuming noisy queries, *etc.*

In most function-computation problems, the major cost of calculating the function is the price of sampling. The sample meaning may change from one model to another but the aspiration to minimize the number of samples is unavoidable in all the models. We mention few specific examples to show how broad this problem is and then we continue our discussion with the general form of the function computation problem.

As the first example, suppose we want to estimate the average height of a population. In this example, the function is the average function and the inputs are people's heights. Since measuring height of the whole population is costly, for estimating the average, we may only measure the height of a sub-population. As a result, we compute a noisy average of the inputs instead of the exact average. As another example, suppose we want to sort the inputs that are real numbers with only pairwise comparisons. In this example, the function outputs a sorted form of the inputs and one can view the pairwise comparisons as samples or *queries* with the goal of minimizing the total number of queries. Hence this example fits to our function computation model. Next, in this broad area of function computation, we present and motivate our model of function computation by relating it to the extensive works that have been done in this area.

*Parts of this paper appeared in [AJO11, DJO+12].

A basic model of multivariate function computation is the *decision-tree complexity* or *worst-case query complexity* [AB09, BN95, Weg87a, San95]. In this model, we find the function value by adaptively choosing the inputs queried while optimizing the maximum number of queries needed to determine the function value. For example, the worst-case query complexity of a Boolean function $xy \vee \overline{x}z$ is 2, as the value of x determines which of y or z needs to be queried to determine the function value. For a survey of decision-tree complexity, please see [BW02].

A multivariate function is *symmetric* if its output remains unchanged under all input permutations. Many functions encountered in engineering and science are symmetric, including *parity*, *threshold*, and *delta*, as well as most statistical measures such as *median*, *mode*, *max*, *etc.* [Weg87b, NW99, Amb05] considered the complexity of symmetric functions.

It has been shown, *e.g.* [DFO10], that the worst-case query complexity of all non-constant symmetric functions is n. On the other hand, average-case query complexity received less attention. Perhaps because it is difficult to find a fixed and robust probabilistic model for the data, or because minimizing the expected depth of a decision tree is more difficult to analyze than minimizing the maximum depth.

Despite getting less attention, *expected* or average-case query complexity for computing a function is more important than the worst-case in most of the problems. For example, when an airline decides to set a price for a ticket, often the expected earning is more of attention than the worst-case or when a gambler wants to decide a strategy, often the focus is on increasing expected gain than worst-case.

[BDCG89, Wan97, BT06] considered the expected query complexity. [AW01] showed that for the expected query complexity under the uniform distribution, quantum algorithms can be exponentially faster than classical algorithms. [KK10] considered the expected query complexity of computing symmetric functions. For the expected query complexity, the optimal query order depends not only on the function, but also on the underlying distribution of variables. [KK10] found an optimal query order for threshold functions of independent but not necessarily identical Bernoulli random variables. In particular, they showed that for threshold functions of independent Bernoulli random variables, the optimal query order does not depend on the precise probabilities of inputs, but only on which is the largest, the second largest, *etc.*

To simplify and extend arguments for finding the optimal query order, [AJO11] defined the expected *verification* query complexity of a function to be the lowest expected number of inputs that need to be revealed to convince an observer of the value of the function. For example, consider the logical OR function $X_1 \vee \ldots \vee X_n$, where each $X_i \sim$ bernoulli(p_i) and independently. To verify that the OR function is 1, it suffices to show that one of the inputs is 1; hence for moderate values of p_i's, the expected number of inputs that need to be revealed is small, whereas verifying that the OR function is 0, requires checking that all inputs are 0, hence all the n inputs must be queried. Note that verification complexity differs from *certificate* complexity [Aar03, BW02, AB09], where all input values are known in advance and can be used to determine the optimal query order.

We show that for all symmetric functions of independent binary inputs, the optimal expected verification and computation complexities are equal. We use this result to simplify the proof of the optimal query complexity of threshold functions presented in [KK10]. We observe that the value of all binary-input symmetric functions depends only on the number of ones, or *weight*, of the input, and use this property to find an optimal query order for all delta functions and for all symmetric functions that are not constant over any three consecutive input weights. We also show that after relaxing any of the symmetry, independence, or binary inputs restrictions, there are functions whose verification complexity is strictly lower than their computation complexity.

The rest of the paper is organized as follows. In Section 2, we formally define the problems of computation and verification. In Section 3, we make a few observations to simplify our analysis. In Section 4, we show that verification method can be used to find the query order for computing threshold and delta functions. In Section 5, we show the equality of verification and computation for general symmetric functions of independent binary inputs. In Section 6, we demonstrate that symmetry, independency, and binary restrictions are all necessary to have equal verification and computation complexities.

2 Notation and formulation

Throughout the paper, except Section 6, we assume that f is a symmetric function of n binary inputs $\mathbf{X} \stackrel{\text{def}}{=} X_1, X_2, \ldots, X_n$, where $X_i \sim \text{bernoulli}(p_i)$ independently, and the p_i's are *known* in advance. Without loss of generality, assume that $1 > p_1 \geq p_2 \geq \ldots \geq p_n > 0$. $[i, j]$ denotes the set of integers between and including i and j. For any set $\mathcal{S}$, $|\mathcal{S}|$ denotes the number of element in $\mathcal{S}$. Let $\bar{p}_i \stackrel{\text{def}}{=} 1 - p_i$.

To compute $f(\mathbf{X})$, we query the inputs sequentially. A *policy* $\mathcal{P}$ is a rule that at any given time, based on prior query outcomes, determines whether querying should stop or continue, and if the latter, which input should be queried next. $\mathcal{P}$ *computes* f, if for all values of $\mathbf{X}$, when $\mathcal{P}$ stops querying, f can be determined.

Let $Q_{\mathcal{P}}(\mathbf{x})$ be the number of inputs a policy $\mathcal{P}$ queries for input $\mathbf{x}$. The expected query complexity of $\mathcal{P}$ is

$$C(\mathcal{P}) \stackrel{\text{def}}{=} \mathbb{E}[Q_{\mathcal{P}}(\mathbf{X})] = \sum_{\mathbf{x}} p(\mathbf{x}) Q_{\mathcal{P}}(\mathbf{x}),$$

and the *computation complexity* of f is

$$C(f) \stackrel{\text{def}}{=} \min_{\mathcal{P}} C(\mathcal{P}) = \min_{\mathcal{P}} \sum_{\mathbf{x}} p(\mathbf{x}) Q_{\mathcal{P}}(\mathbf{x}), \tag{1}$$

where the minimum is taken over all the policies $\mathcal{P}$ computing f. Any policy that computes f with complexity $C(f)$ is an *optimal* computation policy. In general, there might be several optimal computation policies.

Example 1. *Consider the* threshold *function,*

$$\Pi_\theta(w) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } w \geq \theta, \\ 0 & \text{otherwise,} \end{cases}$$

and let $f(x_1, x_2) = \Pi_1(x_1 + x_2)$ determine if at least one of x_1 and x_2 is 1. If X_i is queried first, the expected number of queries is $1 \cdot p_i + 2 \cdot (1 - p_i) = 2 - p_i$, since a policy can stop after querying an input with value one and it has to query the second input if it queries an input with value zero. As a result, the optimal policy should query X_1 first if p_1 is strictly greater than p_2.

In theory, it is possible to express the computation complexity of any function and policy in terms of the input probabilities and optimize the query order. But since the number of policies is exponential in n, this may be computationally inefficient.

An alternative approach was proposed in [AJO11]. Instead of finding an optimal policy to compute a function, they considered the *simpler* problem of finding an optimal policy to verify the function value. They found a class of functions for which the two policies coincide.

In the *verification* of a function f, we are given the value of $f(\mathbf{X})$, and are asked to query the inputs to verify that this is indeed the function value. As with computation, we apply a policy that determines which inputs to query and when to stop so that the function value can be determined. The only difference is that we have a freedom to use different policies for different function values.

It is easy to see that a *verification policy* is just a collection of computation policies, one for each value of f, and the advantage of verification is that for each value of f we can choose a policy that minimizes the expected number of queries for that value of f. The difference between verification and computation complexities is perhaps easier to demonstrate via a non-symmetric function of dependent random variables.

Example 2. *Let $n > 1$ be an integer. Let $\mathbf{e}_i$ be the unit vector in $\mathbb{R}^n$, whose ith component is 1 and all others are 0. Let $\mathbf{X} = \mathbf{e}_i$ with probability $\frac{1}{n}$,* i.e., *one of the n unit vectors with equal probability. Let $f : \{0,1\}^n \to \{1, ..., n\}$ be such that $f(\mathbf{e}_i) = i$. For example, for $n = 3$, $f(100) = 1$, $f(010) = 2$, and $f(001) = 3$, and $Pr(100) = Pr(010) = Pr(001) = 1/3$.*

To compute f, we must find an i such that $X_i = 1$. Therefore, we need to query the inputs till we find the input whose value is 1 or find the $n-1$ inputs whose values are 0. Hence the computation complexity is

$$\frac{1}{n}\left(1+2+\ldots+(n-1)+(n-1)\right) = \frac{(n-1)(n+2)}{2n}.$$

However, for verification we are given $f(\mathbf{X}) = i$ for some i and we can verify $X_i = 1$ by only querying X_i. Hence the verification complexity is 1.

For a more precise definition, the expected query complexity of policy $\mathcal{P}$ when $f(\mathbf{X}) = j$ is

$$C(\mathcal{P}|f(\mathbf{X}) = j) \stackrel{\text{def}}{=} \mathbb{E}\left[Q_{\mathcal{P}}(\mathbf{X})|f(\mathbf{X}) = j\right] = \sum_{\mathbf{x}:f(\mathbf{x})=j} \frac{p(\mathbf{x})Q_{\mathcal{P}}(\mathbf{x})}{\Pr\left(f(\mathbf{X}) = j\right)}.$$

The verification complexity of f when $f(\mathbf{X}) = j$ is the smallest expected number of inputs that need to be queried to verify that $f(\mathbf{X}) = j$, *i.e.,*

$$V_j(f) \stackrel{\text{def}}{=} \min_{\mathcal{P}} C(\mathcal{P}|f(\mathbf{X}) = j) = \min_{\mathcal{P}} \sum_{\mathbf{x}:f(\mathbf{x})=j} \frac{p(\mathbf{x})Q_{\mathcal{P}}(\mathbf{x})}{\Pr\left(f(\mathbf{X}) = j\right)},$$

where the minimum is taken over all policies that verify $f(\mathbf{X}) = j$. Equivalently we can take the minimum over all policies computing f since for the values of f other than j the behavior of the computation policy is not important.

The minimum expected verification query complexity or simply the *verification complexity* of f is

$$V(f) \stackrel{\text{def}}{=} \sum_j \Pr(f(\mathbf{X}) = j)V_j(f) = \sum_j \min_{\mathcal{P}} \sum_{\mathbf{x}:f(\mathbf{x})=j} p(\mathbf{x})Q_{\mathcal{P}}(\mathbf{x}), \tag{2}$$

where for each j, we find a potentially different policy $\mathcal{P}$ minimizing $V_j(f)$.

An optimal verification policy is one whose expected query complexity is $V(f)$. As with its computation counterpart, f may have several optimal verification policies.

In the next section, we make a few observations about computation and verification complexity and in Section 5, we show that for all symmetric functions of independent binary inputs, $V(f) = C(f)$.

3 Preliminary observations

Since computation is one way of verification, or equivalently, a verification policy is a set of computation policies, one for each value of f, the verification complexity is at most the computation complexity.

Observation 3. *For all f, $V(f) \leq C(f)$.*

Proof. Comparing Equations (1) and (2),

$$V(f) = \sum_j \min_{\mathcal{P}} \sum_{\mathbf{x}:f(\mathbf{x})=j} p(\mathbf{x})Q_{\mathcal{P}}(\mathbf{x}) \leq \min_{\mathcal{P}} \sum_{\mathbf{x}} p(\mathbf{x})Q_{\mathcal{P}}(\mathbf{x}) = C(f),$$

since the sum of minimums is at most the minimum of sum. □

Recall that symmetric functions of binary inputs are determined by the input's weight $w \stackrel{\text{def}}{=} w(\mathbf{X}) \stackrel{\text{def}}{=} \sum_{i=1}^n X_i$. With a slight abuse of notation, we use $f(w(\mathbf{X}))$ and $f(\mathbf{X})$ interchangeably. The following observation shows that when a policy computing f stops, the value of f is constant for all possible weights.

Observation 4. *If for inputs $\mathbf{x}$, a policy stops after querying n_0 zeros and n_1 ones, then $w(\mathbf{x})$ can take any value in the contiguous interval $[n_1, n-n_0]$. Furthermore, if the policy computes f, then $f(\mathbf{x})$ is constant for all $\mathbf{x}$ such that $w(\mathbf{x}) \in [n_1, n-n_0]$.*

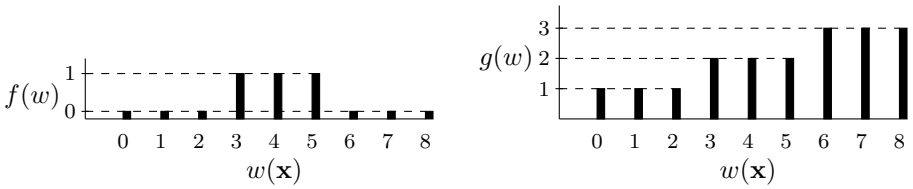

Figure 1: Example of function g

Proof. After querying n_0 zeros and n_1 ones, $\sum_i x_i$ can take any value in $[n_1, n-n_0]$ depending on the values of the unknown inputs. Since when the policy stops, the function value is determined, regardless of the unknown inputs, $f(\mathbf{x})$ must be the same for all inputs with $w(\mathbf{x}) \in [n_1, n-n_0]$. □

Let the *interval indicator function* of f be the function $g : [0, n] \to [1, n+1]$, defined by $g(0) \stackrel{\text{def}}{=} 1$ and the following recursion,

$$g(i+1) - g(i) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } f(i+1) = f(i), \\ 1 & \text{if } f(i+1) \neq f(i). \end{cases}$$

$g(w)$ indicates which peacewise-constant interval of f, $w(\mathbf{X})$ belongs to. Figure 1 demonstrates the relation between f and g by example. Define $\mathcal{I}_j \stackrel{\text{def}}{=} \{w | g(w) = j\}$ to be the j^{th} peacewise-constant interval of the function g. The next observation, lower bounds the number of inputs needed in order to compute g.

Observation 5. *If $w(\mathbf{x}) \in \mathcal{I}_j$ then for any policy $\mathcal{P}$, $Q_{\mathcal{P}}(\mathbf{x}) \geq n - |\mathcal{I}_j| + 1$.*

Proof. From Observation 4, we conclude that $g(x) = j$ for $j \in [n_1, n-n_0]$, where n_0 is the number of queried zeros and n_1 is the number of queried ones. Hence, $[n_1, n-n_0] \subset \mathcal{I}_j$. Therefore, $n - n_0 - n_1 + 1 \leq |\mathcal{I}_j|$. Equality is achieved, when $[n_1, n-n_0] = \mathcal{I}_j$. □

We divide intervals into two types. An interval $\mathcal{I}_j$ is *large* if $|\mathcal{I}_j| \geq \frac{n+1}{2}$ and *small* otherwise. In Section 5, we consider the behavior of optimal verification policies for these two types of intervals separately. The next observation shows that f and g have the same computation complexity.

Observation 6. *For any f, a symmetric function of independent binary random variables,*

$$C(f) = C(g).$$

Proof. For every symmetric function f, the value of g determines the value of f. Therefore, any policy that computes g also computes f. Hence $C(f) \leq C(g)$. Conversely, while the value of f may not determine the value of g, *e.g.,* for the parity function, by Observation 4, when the value of f is determined, $w(\mathbf{X})$ lies in a known interval over which f is constant, and hence g is determined as well. Therefore, a policy that computes f also computes g and $C(g) \leq C(f)$. □

While the same result also holds for verification complexity, only one direction is easy to prove.

Observation 7. *For any f, a symmetric function of independent binary random variables,*

$$V(g) \leq V(f).$$

Proof. Recall that verification policy for f is a collection of computation policies, one for each value of f. Since g determines f, and a verification policy for f is also a verification policy for g, $V(g) \leq V(f)$. □

Our main result, Theorem 15, shows that for all symmetric functions f of independent binary random variables, $V(g) = C(g)$. Combining the above observations, we obtain $C(g) = V(g) \leq V(f) \leq C(f) = C(g)$ and hence,

Corollary 8. *For all symmetric functions f of independent binary random variables,*

$$V(g) = V(f) = C(f) = C(g).$$

In general, verification complexity appears to be easier to determine than computation complexity, and verification complexity of the interval indicator functions seems easier to determine than the verification complexity of symmetric functions.

In the next section we give examples to demonstrate this simplicity. For these examples, we can find the computation policy by only considering the verification one. A generalization of this result is given in Section 5.

4 Simplicity of verification

In this section, we find the optimal computation policies for the threshold and *delta* functions. We start with the threshold function and find a specific optimal verification policy for it. Next we show that there exists a computation policy which follows that specific optimal verification policy. As a result, the computation policy should be optimal since computation complexity is at least verification complexity. Finally we generalize the result to delta functions. We start by a simple lemma about the property of optimal verification policies.

Consider the case when the threshold function $\Pi_\theta(\mathbf{X})$ has value 1. This can be verified only when θ ones have been observed. Therefore, an optimal verification policy should minimize the expected time to observe θ ones. It is intuitive to first query the input with highest probability, *i.e.,* X_1, and then X_2 if it is necessary and so on. The following lemma formalizes this intuition.

Lemma 9. *For $\Pi_\theta(\mathbf{X}) = 1$, an optimal verification policy is to query in the order $X_1, X_2, \dots$ until θ ones are observed and for $\Pi_\theta(\mathbf{X}) = 0$, the order is $X_n, X_{n-1}, \dots$ until $(n - \theta + 1)$ zeros are observed.*

Proof. We prove the optimality when $\Pi_\theta(\mathbf{X}) = 1$. A similar argument holds when $\Pi_\theta(\mathbf{X}) = 0$. It suffices to show that querying X_1 as the first input is optimal since after the first query, the problem reduces to computing another threshold function for the rest of the inputs.

We prove optimality of querying X_1 by induction on (n, θ). If $n < \theta$ the function is trivially zero. If $n = \theta$, all inputs must be queried to verify that $\Pi_\theta(\mathbf{X}) = 1$. Therefore, X_1 can be queried first.

For $\theta \geq 2$, suppose X_k is queried first. If $X_k = 1$, we have to find an optimal verification policy for $n - 1$ inputs and threshold $\theta - 1$, and if $X_k = 0$, we have to find an optimal verification policy for $n - 1$ inputs and threshold θ. By induction hypothesis, for both these cases there is an optimal hypothesis in which X_1 is queried next. This implies that there is an optimal policy for our problem in which the first two observed inputs are X_k and X_1, and since $\theta \geq 2$ at least two inputs should be queried. Thus the order of X_k and X_1 is immaterial and X_1 could be queried first, followed by X_k and no further changes to the policy.

This leaves us with the case when $\theta = 1$. For $n = 1$ it is trivial to ask X_1. For $n = 2$, we proved in Example 1 that X_1 should be queried first. For $n > 2$ and $\theta = 1$, we again use induction to prove X_1 should be queried first. Suppose an optimal policy $\mathcal{P}_1$ that queries $X_k \neq X_1$ first. If $X_k = 1$ then policy $\mathcal{P}_1$ should stop querying the inputs. On the other hand, if $X_k = 0$, then by induction hypothesis, policy $\mathcal{P}_1$ should query X_1 next. Now consider the new policy $\mathcal{P}_2$ which queries X_1 and then X_k if needed, and from the third step onwards, the policy follows the decision of the optimal policy $\mathcal{P}_1$. If $(X_1, X_k) = (0, 0)$ or $(1, 1)$, then $\mathcal{P}_1$ and $\mathcal{P}_2$ query the same number of inputs. Therefore, we only consider the case when $\{X_1, X_k\} = \{0, 1\}$, *i.e.,* one and only one of the X_1 and X_k is 1. Let

$$\alpha \stackrel{\text{def}}{=} \Pr\left(X_1 = 1, X_k = 0 | \{X_1, X_k\} = \{0, 1\}\right) = \frac{p_1 \bar{p}_k}{p_1 \bar{p}_k + \bar{p}_1 p_k} \geq \frac{1}{2}.$$

Observe that $\mathbb{E}\left[Q_{\mathcal{P}_2}(\mathbf{X}) | \{X_1, X_k\} = \{0, 1\}\right] = \alpha + 2\bar{\alpha}$ and $\mathbb{E}\left[Q_{\mathcal{P}_1}(\mathbf{X}) | \{X_1, X_k\} = \{0, 1\}\right] = 2\alpha + \bar{\alpha}$. Since $\alpha \geq \frac{1}{2}$, the expected query complexity of policy $\mathcal{P}_2$ is smaller than or equal to the expected query complexity of policy $\mathcal{P}_1$. Equality only happens when $p_1 = p_k$. As a result, an optimal policy to verify $\Pi_1(\mathbf{X}) = 1$ could query X_1 first. Hence the lemma is proved. □

The next lemma finds an input which can be queried first in an optimal verification policy for threshold function. This lemma helps us to find the optimal computation policy.

Lemma 10. *An optimal verification policy queries X_θ regardless of value of $\Pi_\theta(\mathbf{X})$.*

Proof. In Lemma 9 we showed that if $\Pi_\theta(\mathbf{X}) = 1$, an optimal verification policy queries $X_1, X_2, \ldots$ till finding θ ones. As a result, if $\Pi_\theta(\mathbf{X}) = 1$, an optimal verification policy should query X_θ at some point. Similarly, if $\Pi_\theta(\mathbf{X}) = 0$, an optimal verification policy queries X_θ at some point. □

Lemma 10 shows that if we query X_θ first, we can *track* an optimal verification policy without knowing the function value. After observing the value of X_θ, the threshold function will reduce to a new threshold function over $n-1$ inputs. Therefore, we can continue on tracking the optimal verification policy. Pseudocode of this verification policy is described in Optimal-Policy-Threshold.

let $i = j = \theta$ and $Y = 1$
while θ ones or $n - \theta + 1$ zeros are not queried
    if $Y = 1$
        query X_i, let $Y = X_i$, and decrement i
    else
        query X_j, let $Y = X_j$, and increment j
end while
output Y

Algorithm 1: Optimal-Policy-Threshold

Observe that Optimal-Policy-Threshold is an optimal verification policy for both $\Pi_\theta(\mathbf{X}) = 0$ and $\Pi_\theta(\mathbf{X}) = 1$. This policy does not need to know the function value to proceed and therefore, it is also a computation policy. Since computation complexity is at least verification complexity, Optimal-Policy-Threshold is an optimal computation policy.

In the following we generalize this result to the delta function which is defined as

$$\Delta_\theta(w) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } w = \theta, \\ 0 & \text{otherwise.} \end{cases}$$

When a policy computes $\Delta_\theta(w)$ it also shows us whether $w < \theta$, $w = \theta$ or $w > \theta$. This can be seen by Observation 4. We consider the problem of verification for these cases separately. Using the results obtained for threshold functions, we obtain the following optimal verification policies.

- $w = \theta$: In this case all inputs must be queried to verify that the weight is exactly θ.
- $w < \theta$: This problem is identical to the threshold problem with θ as the threshold value. We know that an optimal verification policy is to query $X_n, X_{n-1}, \ldots$ until $n - \theta + 1$ zeros are observed.
- $w > \theta$: This problem is identical to the threshold problem with $\theta + 1$ as the threshold value. We know that an optimal verification policy is to query $X_1, X_2, \ldots$ until $\theta + 1$ ones are observed.

By comparing these optimal verification policies to optimal verification policies for the threshold function we can easily show that Optimal-Policy-Threshold can be used for optimal computation policy for delta function with only changing the stopping criterion. The new stopping criteria is when $\theta+1$ ones or $n-\theta+1$ zeros are observed, or all inputs have been queried. For the sake of completeness, the Pseudocode for this policy is given in Optimal-Policy-Delta.

let $i = j = \theta$ and $Y = 1$
while $\theta + 1$ ones or $n - \theta + 1$ zeros are not queried
    if all the inputs are queried
        output 1 and stop
    if $Y = 1$
        query X_i, let $Y = X_i$, and decrement i
    else
        query X_j, let $Y = X_j$, and increment j
end while
output 0

Algorithm 2: OPTIMAL-POLICY-DELTA

5 Equality of verification and computation complexities for symmetric functions of independent binary inputs

In this section we consider general symmetric functions of independent binary inputs. Similar to Section 4, we show that verification and computation complexities of such functions coincide. In Theorem 15 we show that there is a set of inputs that can be queried first regardless of the function (output) value. This result will yield in an optimal computation policy that is also an optimal verification policy for all the function values.

The proof of Theorem 15 is slightly involved and Figure 2 illustrates its key components. Recall the definitions of large and small intervals from Section 3. In Lemma 13, we consider the optimal verification policy when the input weight belongs to a large interval and we find some inputs of which one can be queried first in an optimal verification policy. Conversely, when the weight of the inputs belongs to a small interval, Lemma 14 finds some inputs any of which can be queried first in an optimal verification policy. A combination of these two lemmas finds an input that can be queried first in an optimal verification policy independent of the function value. Lemmas 11 and 12 will be the main tools to prove Lemmas 13 and 14.

By definition, a policy starts by querying a fixed input X_i and the next input to query is a function of the value of X_i. An optimal policy is called *second-input-fixed* if the second input queried is some fixed X_j independent of the value of X_i. On the other hand, a policy may query X_i first and then X_j or X_k next depending on whether $X_i = 0$ or $X_i = 1$. Such a policy is called *second-input-varies*.

Second Input Fixed The next result is about second-input-fixed optimal policies. Suppose there is a function for which there exists a second-input-fixed optimal policy, that queries some two inputs X_i and X_j. Using the fact that the input probabilities are sorted and the function is symmetric, we show that for any index k between i and j, there exists an optimal policy that queries X_k first.

Lemma 11. *Let $\mathcal{P}_{opt}$ be a second-input-fixed optimal policy (verification or computation) that queries X_i and X_j as the first two inputs. Then for any k between i and j (inclusive), there exists an optimal policy that queries X_k first.*

Proof. We first note that since the function is symmetric and inputs are independent, when $X_i + X_j = 1$, the policy which is optimal for the case $X_i = 0$ and $X_j = 1$ is also optimal for $X_i = 1$ and $X_j = 0$. Let $\mathcal{P}_{opt,2}$ denote this policy. Let $\mathcal{P}_{opt,1}$, and $\mathcal{P}_{opt,3}$ denote optimal policies when $X_i + X_j = 0$, and $X_i + X_j = 2$ respectively. Figure 3 gives a schematic of the optimal policy for computing or verifying the value of the function in this setting.

Without loss of generality, we assume that $i < j$, and let $i < k < j$. We will consider four policies, each of which queries X_k first, and show that at least one of them is as good as $\mathcal{P}_{opt}$. We note that which of these is optimal might be depend on the function itself. We now briefly describe the functioning of these policies (See illustration in Figure 4). After querying X_k, depending on whether its values is 0 or 1, we query either X_i or X_j. This has four possible choices. After these two queries, the policy follows the steps

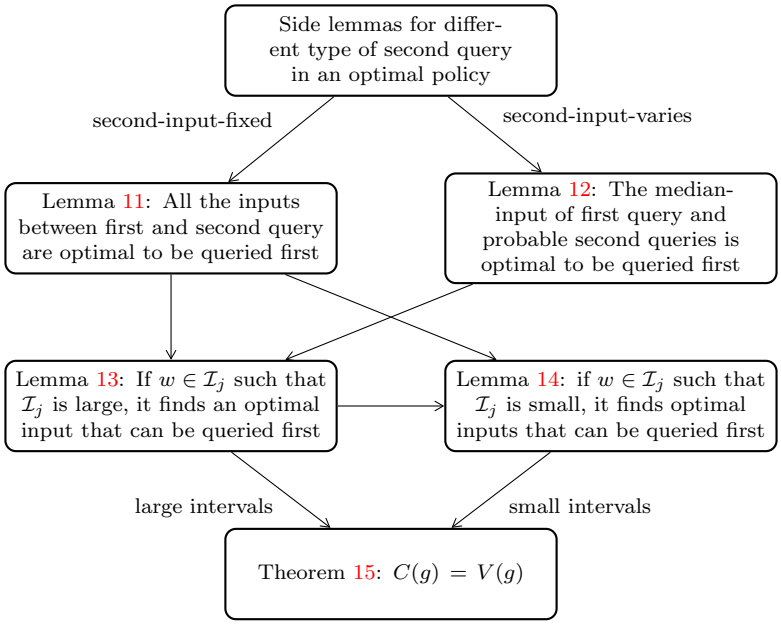

Figure 2: Proof scheme for Theorem 15

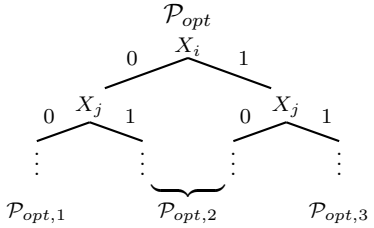

Figure 3: Second-input-fixed policy

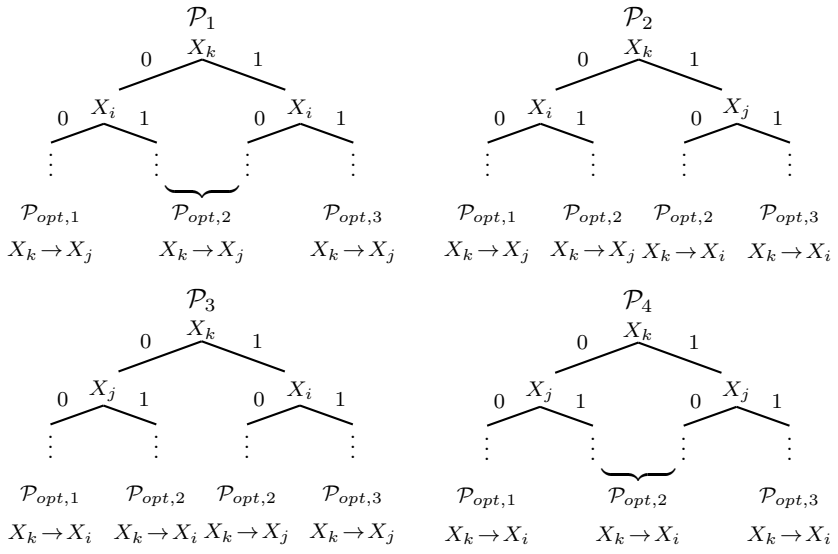

Figure 4: Candidates for the optimal policy if second input is fixed

of $\mathcal{P}_{opt}$ (imagining X_k as one of the X_i or X_j which was not queried). When $\mathcal{P}_{opt}$ requires querying X_k (in the original policy), we simply query X_i or X_j (the one that was not queried). We next describe this in a little more detail.

The first policy queries X_k and then X_i. It then follows $\mathcal{P}_{opt}$ and queries X_j when $\mathcal{P}_{opt}$ required querying X_k. In the second policy X_k is queried first, and depending on its value we query X_i or X_j and follow $\mathcal{P}_{opt}$. When $\mathcal{P}_{opt}$ required querying X_k, we query the input among X_i and X_j that has not been queried yet. The remaining two policies are similarly defined, taking care of the other cases.

We show that the expected query complexity of at least one of the four policies defined here is at most the query complexity of $\mathcal{P}_{opt}$, namely one of these policies is as good as $\mathcal{P}_{opt}$.

Using linearity of expectations, the complexity can be written as

$$\begin{aligned} C(\mathcal{P}_{opt}) = \bar{p}_i\bar{p}_j(a_{00}\bar{p}_k + b_{00}p_k) + \bar{p}_ip_j(a_{01}\bar{p}_k + b_{01}p_k) + \\ p_i\bar{p}_j(a_{01}\bar{p}_k + b_{01}p_k) + p_ip_j(a_{11}\bar{p}_k + b_{11}p_k), \end{aligned}$$

where $a_{r,t}$ and $b_{r,t}$ are non-negative numbers depending on the structure of $\mathcal{P}_{opt}$ (which in turn depends on the output function), for $r, t \in \{0, 1\}$, independent of p_i, p_j and p_k. The complexity of the four potential policies can therefore be written as,

$$\begin{aligned} C(\mathcal{P}_1) &= \bar{p}_k\bar{p}_i(a_{00}\bar{p}_j + b_{00}p_j) + \bar{p}_kp_i(a_{01}\bar{p}_j + b_{01}p_j) + \\ &\quad p_k\bar{p}_i(a_{01}\bar{p}_j + b_{01}p_j) + p_kp_i(a_{11}\bar{p}_j + b_{11}p_j), \\ C(\mathcal{P}_2) &= \bar{p}_k\bar{p}_i(a_{00}\bar{p}_j + b_{00}p_j) + \bar{p}_kp_i(a_{01}\bar{p}_j + b_{01}p_j) + \\ &\quad p_k\bar{p}_j(a_{01}\bar{p}_i + b_{01}p_i) + p_kp_j(a_{11}\bar{p}_i + b_{11}p_i), \\ C(\mathcal{P}_3) &= \bar{p}_k\bar{p}_j(a_{00}\bar{p}_i + b_{00}p_i) + \bar{p}_kp_j(a_{01}\bar{p}_i + b_{01}p_i) + \\ &\quad p_k\bar{p}_i(a_{01}\bar{p}_j + b_{01}p_j) + p_kp_i(a_{11}\bar{p}_j + b_{11}p_j), \\ C(\mathcal{P}_4) &= \bar{p}_k\bar{p}_j(a_{00}\bar{p}_i + b_{00}p_i) + \bar{p}_kp_j(a_{01}\bar{p}_i + b_{01}p_i) + \\ &\quad p_k\bar{p}_j(a_{01}\bar{p}_i + b_{01}p_i) + p_kp_j(a_{11}\bar{p}_i + b_{11}p_i). \end{aligned}$$

In order to compare these policies with the optimal policy, we subtract $\mathcal{P}_{opt}$ from each of them. Upon

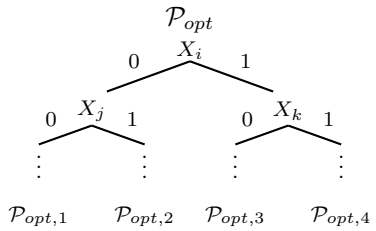

Figure 5: Second-input-varies policy

simplification, this yields

$$\begin{aligned}
C(\mathcal{P}_1) - C(\mathcal{P}_{opt}) &= \bar{p}_i(p_j - p_k)(b_{00} - a_{01}) + p_i(p_j - p_k)(b_{01} - a_{11}),\\
C(\mathcal{P}_2) - C(\mathcal{P}_{opt}) &= \bar{p}_i(p_j - p_k)(b_{00} - a_{01}) + p_j(p_i - p_k)(b_{01} - a_{11}),\\
C(\mathcal{P}_3) - C(\mathcal{P}_{opt}) &= \bar{p}_j(p_i - p_k)(b_{00} - a_{01}) + p_i(p_j - p_k)(b_{01} - a_{11}),\\
C(\mathcal{P}_4) - C(\mathcal{P}_{opt}) &= \bar{p}_j(p_i - p_k)(b_{00} - a_{01}) + p_j(p_i - p_k)(b_{01} - a_{11}).
\end{aligned}$$

By assumption, input probabilities are sorted in decreasing order, and hence $p_i \geq p_k \geq p_j$. If any of p_i, p_j, and p_k are equal, the result follows trivially. Assume that they are all different. In the four equations above, observe that the coefficients multiplied by $(b_{00} - a_{01})$ and $(b_{01} - a_{11})$ take all possible negative and positive signs. Therefore, either all the equations are zero or at least one of them is negative. Hence, at least one of the four policies performs at least as well as the optimal policy. □

Second Input Varies We now consider second-input-varies policies. Consider an optimal policy that queries X_i first and then X_j or X_k depending on whether $X_i = 0$ or $X_i = 1$ respectively. For the inputs X_i, X_j, X_k, the median index is defined as the median of $\{i, j, k\}$. We call the corresponding input as the *median-input*. We now show that there is an optimal policy that queries the median of these inputs first.

Lemma 12. *For the second-input-varies policy defined above, there exists an optimal policy that queries the median-input of* $\{X_i, X_j, X_k\}$ *first.*

Proof. If $j < i < k$ or $k < i < j$, then by definition, the optimal policy first queries the median-input. We consider the case when $i > \max\{j, k\}$. The case when $i < \min\{j, k\}$ is similar.
Figure 5 describes an optimal policy $\mathcal{P}_{opt}$. As in the proof of Lemma 11,

$$\begin{aligned}
C(\mathcal{P}_{opt}) = \bar{p}_i\bar{p}_j(\bar{p}_k a_{00} + p_k b_{00}) + \bar{p}_i p_j(\bar{p}_k a_{01} + p_k b_{01}) +\\
p_i\bar{p}_k(\bar{p}_j a_{10} + p_j b_{10}) + p_i p_k(\bar{p}_j a_{11} + p_j b_{11}).
\end{aligned}$$

Since $\mathcal{P}_{opt}$ is optimal, changing $\mathcal{P}_{opt,2}$ to $\mathcal{P}_{opt,3}$ in Figure 5 should not decrease the complexity. As a result,

$$\bar{p}_k a_{01} + p_k b_{01} \leq \bar{p}_k a_{10} + p_k b_{10}. \tag{3}$$

Similarly,

$$\bar{p}_j a_{10} + p_j b_{10} \leq \bar{p}_j a_{01} + p_j b_{01}. \tag{4}$$

We consider the two possible orderings of j and k and for each we show that there exists an optimal policy that queries the median input first.
Case 1 $(j < k < i)$ We consider the policies in Figure 6 and show that at least one of them is as good as $\mathcal{P}_{opt}$. Note that both of these policies query the median-input first. By the linearity of expectation, the expected query complexity of the two policies in Figure 6 are

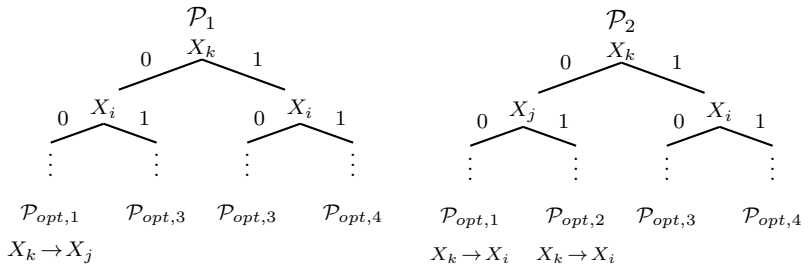

Figure 6: Candidates for the optimal policy if second input varies, $j < k$

$$
\begin{aligned}
C(\mathcal{P}_1) &= \bar{p}_k\bar{p}_i(\bar{p}_j a_{00} + p_j b_{00}) + \bar{p}_k p_i(\bar{p}_j a_{10} + p_j b_{10}) + \\
&\quad p_k\bar{p}_i(\bar{p}_j a_{10} + p_j b_{10}) + p_k p_i(\bar{p}_j a_{11} + p_j b_{11}), \\
C(\mathcal{P}_2) &= \bar{p}_k\bar{p}_j(\bar{p}_i a_{00} + p_i b_{00}) + \bar{p}_k p_j(\bar{p}_i a_{01} + p_i b_{01}) + \\
&\quad p_k\bar{p}_i(\bar{p}_j a_{10} + p_j b_{10}) + p_k p_i(\bar{p}_j a_{11} + p_j b_{11}).
\end{aligned}
$$

either $C(\mathcal{P}_1) \leq C(\mathcal{P}_{opt})$ or $C(\mathcal{P}_2) \leq C(\mathcal{P}_{opt})$.
If $C(\mathcal{P}_1) > C(\mathcal{P}_{opt})$, then

$$p_j\bar{p}_k\bar{p}_i b_{00} + \bar{p}_j p_k\bar{p}_i a_{10} + p_j p_k\bar{p}_i b_{10} > p_j\bar{p}_k\bar{p}_i a_{01} + \bar{p}_j p_k\bar{p}_i b_{00} + p_j p_k\bar{p}_i b_{01}.$$

Applying (4),

$$p_j\bar{p}_k\bar{p}_i b_{00} + \bar{p}_j p_k\bar{p}_i a_{01} > p_j\bar{p}_k\bar{p}_i a_{01} + \bar{p}_j p_k\bar{p}_i b_{00}.$$

Since $p_j > p_k$, we have

$$b_{00} > a_{01}. \tag{5}$$

If $C(\mathcal{P}_2) > C(\mathcal{P}_{opt})$,

$$
\begin{aligned}
&\bar{p}_j\bar{p}_k p_i b_{00} + p_j\bar{p}_k p_i b_{01} + \bar{p}_j p_k\bar{p}_i a_{10} + p_j p_k\bar{p}_i b_{10} > \\
&\qquad \bar{p}_j\bar{p}_k p_i a_{10} + p_j\bar{p}_k p_i b_{10} + \bar{p}_j p_k\bar{p}_i b_{00} + p_j p_k\bar{p}_i b_{01},
\end{aligned}
$$

which can be rewritten as

$$\bar{p}_j\bar{p}_k p_i b_{00} + p_j\bar{p}_k p_i b_{01} + (p_k\bar{p}_i - p_i\bar{p}_k)(\bar{p}_j a_{10} + p_j b_{10}) > \bar{p}_j p_k\bar{p}_i b_{00} + p_j p_k\bar{p}_i b_{01}.$$

Since $p_k\bar{p}_i - p_i\bar{p}_k \geq 0$, using (4) this expression simplifies to

$$\bar{p}_j\bar{p}_k p_i b_{00} + p_j\bar{p}_k p_i b_{01} + (p_k\bar{p}_i - p_i\bar{p}_k)(\bar{p}_j a_{01} + p_j b_{01}) > \bar{p}_j p_k\bar{p}_i b_{00} + p_j p_k\bar{p}_i b_{01}.$$

Grouping terms, this yields

$$(p_k\bar{p}_i - p_i\bar{p}_k)\bar{p}_j a_{01} > (p_k\bar{p}_i - p_i\bar{p}_k)\bar{p}_j b_{00}.$$

Since $p_k\bar{p}_i - p_i\bar{p}_k \geq 0$, this implies $a_{01} > b_{00}$, contradicting (5).
Case 2 ($k < j < i$) Inequalities (3) and (4) still hold. As before we have two policies in Figure 7 and show that at least one of them is as good as our optimal policy. This time we replace $\mathcal{P}_{opt,3}$ by $\mathcal{P}_{opt,2}$ as one of the choices. By the linearity of expectation, the expected query complexity of the two policies are

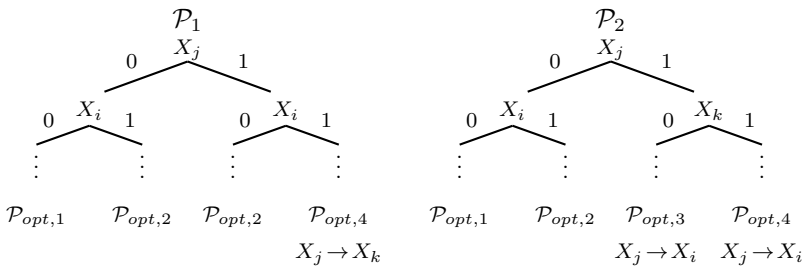

Figure 7: Candidates for the optimal policy if second input varies, $j > k$

$$\begin{aligned}
C(\mathcal{P}_1) &= \bar{p}_j\bar{p}_i(\bar{p}_k a_{00} + p_k b_{00}) + \bar{p}_j p_i(\bar{p}_k a_{01} + p_k b_{01}) + \\
&\quad p_j\bar{p}_i(\bar{p}_k a_{01} + p_k b_{01}) + p_j p_i(\bar{p}_k a_{11} + p_k b_{11}), \\
C(\mathcal{P}_2) &= \bar{p}_j\bar{p}_i(\bar{p}_k a_{00} + p_k b_{00}) + \bar{p}_j p_i(\bar{p}_k a_{01} + p_k b_{01}) + \\
&\quad p_j\bar{p}_k(\bar{p}_i a_{10} + p_i b_{10}) + p_j p_k(\bar{p}_i a_{11} + p_i b_{11}).
\end{aligned}$$

If $C(\mathcal{P}_1) > C(\mathcal{P}_{opt})$,

$$\begin{aligned}
&\bar{p}_j\bar{p}_k p_i a_{01} + \bar{p}_j p_k p_i b_{01} + p_j\bar{p}_k p_i a_{11} > \bar{p}_j\bar{p}_k p_i a_{10} + \bar{p}_j p_k p_i a_{11} + p_j\bar{p}_k p_i b_{10} \\
\Rightarrow\ &\bar{p}_j\bar{p}_k a_{01} + \bar{p}_j p_k b_{01} - \bar{p}_j\bar{p}_k a_{10} > \bar{p}_j p_k a_{11} + p_j\bar{p}_k b_{10} - p_j\bar{p}_k a_{11}.
\end{aligned}$$

Subtracting $\bar{p}_j p_k b_{10}$ from both sides,

$$\bar{p}_j\left(\bar{p}_k a_{01} + p_k b_{01} - \bar{p}_k a_{10} - p_k b_{10}\right) > -\left(\bar{p}_j p_k - p_j\bar{p}_k\right)\left(b_{10} - a_{11}\right). \tag{6}$$

Let $E = \bar{p}_k a_{10} + p_k b_{10} - \bar{p}_k a_{01} - p_k b_{01}$. (3) shows that $E \geq 0$. Observe that $\bar{p}_j p_k - p_j\bar{p}_k = p_k - p_j > 0$. Therefore, (6) can be rewritten as

$$(p_k - p_j)(b_{10} - a_{11}) > \bar{p}_j \cdot E. \tag{7}$$

If $C(\mathcal{P}_2) > C(\mathcal{P}_{opt})$,

$$\begin{aligned}
&\bar{p}_j\bar{p}_k p_i a_{01} + \bar{p}_j p_k p_i b_{01} + p_j\bar{p}_k\bar{p}_i a_{10} + p_j p_k\bar{p}_i a_{11} > \\
&\qquad \bar{p}_j\bar{p}_k p_i a_{10} + \bar{p}_j p_k p_i a_{11} + p_j\bar{p}_k\bar{p}_i a_{01} + p_j p_k\bar{p}_i b_{01},
\end{aligned}$$

which can be rewritten as

$$\left(p_j\bar{p}_i - p_i\bar{p}_j\right)\left(\bar{p}_k a_{10} - \bar{p}_k a_{01} - p_k b_{01}\right) > -p_k(p_j - p_i)a_{11}.$$

Observe that $p_j\bar{p}_i - p_i\bar{p}_j = p_j - p_i > 0$. Adding $(p_j - p_i)p_k b_{10}$ to both sides and substituting $E = \bar{p}_k a_{10} + p_k b_{10} - \bar{p}_k a_{01} - p_k b_{01}$,

$$E > p_k(b_{10} - a_{11}). \tag{8}$$

Since $p_k > p_j$, combining (7) and (8) gives

$$\begin{aligned}
E &> \frac{p_k\bar{p}_j}{p_k - p_j}E \\
&= \frac{p_k - p_j p_k}{p_k - p_j}E \\
&> E,
\end{aligned}$$

a contradiction proving the lemma. □

Recall that we defined intervals to be either large or small in Section 3. In Lemmas 13 and 14 we consider the first input queried for optimal policies, for the cases when the input weight lies in large and small interval respectively.

Lemma 13. *Let $w(\mathbf{X})$ be in a large interval with size L. There exists an optimal policy that queries some X_i first where $i \in [n-L+1, L]$, and verifies the value of $g(w)$.*

Proof. Suppose $w(\mathbf{X}) \in \mathcal{I}_m$ where $\mathcal{I}_m$ is a large interval. Let $\mathcal{L}_{n,L} = \{X_{n-L+1}, \ldots, X_L\}$. Our goal is to show that there exists an optimal verification policy that queries some $X_i \in \mathcal{L}_{n,L}$ first.

The proof is by induction on n.

We first consider some corner cases. Consider the case when $n \in \mathcal{I}_m$. Then verifying the value of $g(w)$ is same as verifying $w(\mathbf{X}) \geq n-L+1$ or $\Pi_{n-L+1}(w) = 1$. From Lemma 10 on threshold functions, X_{n-L+1} can be queried first in an optimal verification policy and $X_{n-L+1} \in \mathcal{L}_{n,L}$. Hence the lemma holds for this corner case. The case of $0 \in \mathcal{I}_m$ follows similarly for verifying that a threshold function is 0.

We now prove the result when 0 and n are not in $\mathcal{I}_m$.

Assume the induction hypothesis that the lemma holds for any function g' with $n' = n-1$ inputs. Observation 5 shows that at least $n-L+1$ inputs must be queried. Since $0 \notin \mathcal{I}_m$ and $n \notin \mathcal{I}_m$, L is smaller than n and at least two inputs need to be queried.

Let $\mathcal{P}_{opt}$ be an optimal policy that queries X_i first. If $X_i \in \mathcal{L}_{n,L}$ then thre is nothing to prove. If $X_i \notin \mathcal{L}_{n,L}$, then either $i > L$ or $i < n-L+1$. We consider the case, when $i > L$ and the case $i < n-L+1$ follows similarly.

Let $\mathbf{X}'$ be all the inputs except X_i. Verifying $w(\mathbf{X}) \in \mathcal{I}_m$ is equivalent to verifying $w(\mathbf{X}') + X_i \in \mathcal{I}_m$. Since 0 and n are not in $\mathcal{I}_m$, after querying X_i, verifying $w(\mathbf{X}') + X_i \in \mathcal{I}_m$ is same as verifying $w(\mathbf{X}') \in \mathcal{I}'_m$ for some $\mathcal{I}'_m$ such that $|\mathcal{I}'_m| = L$.

By induction, there is an optimal policy $\mathcal{P}'_{opt}$ that verifies $w(\mathbf{X}) \in \mathcal{I}_m$ and queries X_i first and then based on its value, it chooses one of the inputs inside $\mathcal{L}_{n-1,L}$ for the second query. Either the policy $\mathcal{P}'_{opt}$ is second-input-fixed or second-input-varies and we consider them separately.

If $\mathcal{P}'_{opt}$ is second-input-fixed, then suppose it queries X_j as the second input such that $X_j \in \mathcal{L}_{n-1,L}$. By Lemma 11, any inputa among $\{X_j, \ldots, X_i\}$ can be queried first in an optimal policy and $\{X_j, \ldots, X_i\} \bigcap \mathcal{L}_{n,L} \neq \emptyset$. If the policy is second-input-varies, then suppose it queries X_j if $X_i = 0$, and X_k if $X_i = 1$, such that $X_j, X_k \in \mathcal{L}_{n-1,L}$. Then, by Lemma 12, there is an optimal policy that queries the median-input of $\{X_i, X_j, X_k\}$, which is in $\mathcal{L}_{n,L}$, first. Therefore, the lemma is proved for $i > L$ and similar proof holds for $i < n-L+1$. □

The next lemma considers the case when the weight belongs to a small interval.

Lemma 14. *Suppose $w(\mathbf{X})$ belongs to a small interval of size ℓ, then for each input in $\{X_\ell, \ldots, X_{n-\ell+1}\}$, there is an optimal policy that queries that input first.*

Proof. The proof is by an induction on n and is similar to the proof of Lemma 13. We explain the general scheme of the proof.

Consider an optimal policy that queries X_i first. The value of i can be in one three distinct sets, $[1, \ell-1]$, $[\ell, n-\ell+1]$, and $[n-\ell+2, n]$. For all these possibilities, using Lemmas 11, 13 and induction hypothesis, we can show that $\{X_\ell, \ldots, X_{n-\ell+1}\}$ are eligible choices to be queried first in some optimal policy. □

The following theorem is the main contribution of the paper. It states that for all symmetric functions of independent binary inputs, there exists an optimal computation policy that is also an optimal verification policy.

Theorem 15. $C(g) = V(g)$.

Proof. Observe that all symmetric functions have at most one large interval, except the threshold function $\Pi_{\frac{n+1}{2}}(x)$ for odd n. The theorem is proved for this function in Section 4. In the remainder of the proof we assume that there is at most one large interval.

Let $\mathcal{A}_j$ be the set of inputs that can be queried first in the optimal verification policy when $w \in \mathcal{I}_j$. It suffices to show that $\bigcap_j \mathcal{A}_j \neq \emptyset$ since it implies that there is an input that can be queried first in an optimal verification policy regardless of the function value. After the first query, the similar argument can be used for the rest of the inputs and we can *track* the optimal verification policies without knowing function value.

Let $\ell_j \stackrel{\text{def}}{=} |\mathcal{I}_j|$ and k be the index of the longest interval. If there is more than one longest interval, choose one of them arbitrarily. We consider the two cases when $\ell_k < \frac{n+1}{2}$ and $\ell_k \geq \frac{n+1}{2}$ separately. For both the cases, we show that $\bigcap_j \mathcal{A}_j \neq \emptyset$.

- When $\ell_k < (n+1)/2$, then $\mathcal{I}_k$ is small interval. By Lemma 14, for all j,

  $$\{X_{\ell_j}, \ldots, X_{n-\ell_j+1}\} \subseteq \mathcal{A}_j.$$

  Therefore, $\{X_{\ell_k}, \ldots, X_{n-\ell_k+1}\} \subseteq A_j$ and $\bigcap_j \mathcal{A}_j \neq \emptyset$.

- When $\ell_k \geq (n+1)/2$, hence $\mathcal{I}_k$ is large interval. By Lemma 14, $\forall j \neq k$,

  $$\{X_{\ell_j}, \ldots, X_{n-\ell_j+1}\} \subseteq \mathcal{A}_j.$$

  Note that,

  $$\{X_{n-\ell_k+1}, \ldots, X_{\ell_k}\} \subseteq \{X_{\ell_j}, \ldots, X_{n-\ell_j+1}\}.$$

  Therefore, $\forall j \neq k, \{X_{n-\ell_k+1}, \ldots, X_{\ell_k}\} \subseteq \mathcal{A}_j$. By Lemma 13,

  $$\{X_{n-\ell_k+1}, \ldots, X_{\ell_k}\} \bigcap \mathcal{A}_k \neq \emptyset.$$

  Hence $\bigcap_j \mathcal{A}_j \neq \emptyset$.

Therefore, $\cap \mathcal{A}_j$ is not empty. □

From the proof of Theorem 15, we have the following observation. Let $\ell_{\max} \stackrel{\text{def}}{=} \max_j |\mathcal{I}_j|$.

Observation 16. *If $\ell_{\max} < (n+1)/2$, an optimal computation policy queries all the inputs $X_{\ell_{\max}}, \ldots, X_{n-\ell_{\max}+1}$ and if $\ell_{\max} = (n+1)/2$, it queries $X_{\ell_{\max}}$.*

Observation 16 can be used to find an optimal computation policy for a function with small $\ell_{\max}$. In particular, if $\ell_{\max}$ is small, using Observation 16, we reduce the number of inputs from n to $2\ell_{\max} - 2$. For a smaller number of inputs, it is possible to find the optimal computation policy using an exhaustive search. In the following, we give an example to show how this result can be used in order to find an optimal computation policy.

Example 17. *In this example, we find the optimal computation policy for functions with $\ell_{max} \leq 2$. Observation 16 shows that the optimal computation policy should query all the inputs $X_2, \ldots, X_{n-1}$. As a result, only X_1 and X_n are left to be queried and the optimal computation policy for any symmetric function with two inputs is trivial.*

6 Functions with different verification and computation complexities

Section 5 shows that computation and verification complexities are same for symmetric functions of independent binary inputs. We show that the symmetry, independence, and binary restrictions are necessary, and relaxing any of them may result in functions whose verification complexity is strictly lower than their computation complexity.

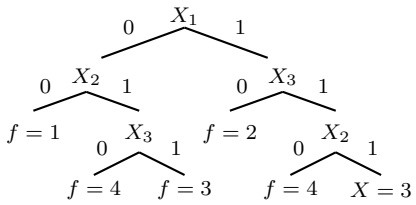

Figure 8: Optimal computation policy for Example 18

6.1 Non-symmetric Functions

The following example is a non-symmetric function of independent binary inputs that has different verification and computation complexities.

Example 18. *Let X_1, X_2, and X_3 be independent* bernoulli($\frac{1}{2}$) *random variables. Consider the non-symmetric function $f(\mathbf{x})$ defined in Table 1. We show that its verification complexity is,*

$$V(f) = \sum_{j=1}^{4} V_j(f) p(f(\mathbf{X}) = j) = \frac{2}{4} + \frac{2}{4} + \frac{2}{4} + \frac{3}{4} = \frac{9}{4}.$$

$x_1x_2x_3$	000	001	100	110	011	111	010	101
$f(\mathbf{x})$	1	1	2	2	3	3	4	4

Table 1: Function in Example 18

The first three terms correspond to function values 1, 2, and 3, each occurring with probability 1/4 *and verifying them requires to query exactly two inputs. The fourth term corresponds to function value 4. In that case, querying all the three inputs is necessary to verify the function value.*

On the other hand, we show that the computation complexity is $\frac{5}{2}$. Suppose we start with querying X_1 as the first input. Querying X_2 or X_3 has a similar result. Based on the X_1 value, the function can be rewritten as in Table 2.

$x_1 = 0$

x_2x_3	00	01	11	10
$f(\mathbf{x})$	1	1	3	4

$x_1 = 1$

x_2x_3	00	10	11	01
$f(\mathbf{x})$	2	2	3	4

Table 2: Induced functions in Example 18

The optimal computation policy that queries X_1 first is shown in Figure 8. A simple calculation shows that the expected query complexity is $\frac{5}{2}$. As a result, computation and verification complexity are different for this example.

Example 18 has a non-binary output. Next, we consider a binary-input and output function where the inputs are independent and the verification and computation complexities are different.

Example 19. *Let X_1, X_2, and X_3 be independent* bernoulli($\frac{1}{2}$) *and X_4 be* bernoulli($\frac{1}{10}$). *Consider the non-symmetric function $f(\mathbf{x})$ defined in Table 3*

With exhaustive search, we can show that if $f = 0$, then an optimal verification policy should *query X_4 first, however if $f = 1$ querying X_3 before X_4 results in a smaller expected query complexity. As a result, different function values imply different set of inputs to be queried first in an optimal verification policy. Therefore, computation and verification complexity are different for this example.*

$x_1x_2x_3x_4$	0000	1000	0100	1100	0010	1010	0110	1110
$f(\mathbf{x})$	0	0	1	1	0	1	0	1
$x_1x_2x_3x_4$	0001	1001	0101	1101	0011	1011	0111	1111
$f(\mathbf{x})$	1	0	1	1	1	0	1	1

Table 3: Function in Example 19

6.2 Functions with dependent inputs

The next example is of a symmetric binary-input function that with dependent inputs has different verification and computation complexities.

Example 20. *Consider the symmetric binary function and input probabilities shown in Table 4, where ϵ is a small positive constant so we can ignore its effect on our calculation.*

$f(\mathbf{x})=1$	$x_1x_2x_3x_4$	1100	1010	0110	0101	0011	1001
	$p(\mathbf{x})$	ϵ	ϵ	q	q	q	$3q$
$f(\mathbf{x})=1$	$x_1x_2x_3x_4$	1110	1101	1011	0111	1111	
	$p(\mathbf{x})$	ϵ	ϵ	ϵ	ϵ	ϵ	
$f(\mathbf{x})=0$	$x_1x_2x_3x_4$	0000	1000	0100	0010	0001	
	$p(\mathbf{x})$	ϵ	ϵ	ϵ	ϵ	q	

Table 4: Function and input probabilities in Example 20

If $f(\mathbf{X})=0$, in an optimal verification policy, all the inputs other than X_4 are eligible to be queried first. However, if $f(\mathbf{X})=1$, We can show that the first two steps of an optimal verification policy have to follow Figure 9. Therefore, based on the function value, an optimal verification policy should query different set of inputs first. Hence the verification and computation complexities are different.

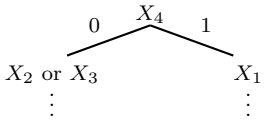

Figure 9: Optimal verification policy in Example 20 when $f(\mathbf{X})=1$

6.3 Non-binary inputs

Our last example is a symmetric function of ternary independent inputs where the verification and computation complexities are different.

Example 21. *Let four independent ternary random variables be distributed over ternary alphabet according to Table 5. And consider the symmetric function of the variable sum*

$$f(\mathbf{x})=\begin{cases}0 & \text{if } \sum_{i=1}^{4} x_i \le 2,\\ 1 & \text{if } 3 \le \sum_{i=1}^{4} x_i \le 5,\\ 0 & \text{if } 6 \le \sum_{i=1}^{4} x_i.\end{cases}$$

The minimum number of inputs to verify $f(\mathbf{X})=0$, namely $\sum_{i=1}^{4} X_i \le 2$, is 3 and the optimal verification policy will query X_2, X_3 or X_4 as a first input but not X_1, since X_1 has the smallest probability of zero among the inputs. Similarly, If $\sum_{i=1}^{4} x_i \ge 6$ the optimal verification policy will query X_1, X_3 or X_4 as a first

	0	1	2
x_1	2ϵ	$\frac{1}{2}-\epsilon$	$\frac{1}{2}-\epsilon$
x_2	$\frac{1}{2}-\epsilon$	$\frac{1}{2}-\epsilon$	2ϵ
x_3	$\frac{1}{2}-\epsilon$	2ϵ	$\frac{1}{2}-\epsilon$
x_4	0.1	0.7	0.2

Table 5: Input probabilities in Example 21

input but not X_2. With exhaustive search, we can show that if $3 \leq \sum_{i=1}^{4} x_i \leq 5$ the optimal verification policy should query X_1 first. Since the first queries for different function values does not intersect, verification and computation complexities are different.

References

[Aar03] Scott Aaronson. Quantum certificate complexity. In *18th Annual IEEE Conference on Computational Complexity (Complexity 2003), 7-10 July 2003, Aarhus, Denmark*, pages 171–178. IEEE, 2003.

[AB09] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[AJO11] Jayadev Acharya, Ashkan Jafarpour, and Alon Orlitsky. Expected query complexity of symmetric boolean functions. In *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on*, pages 26–29. IEEE, 2011.

[Amb05] Andris Ambainis. Polynomial degree and lower bounds in quantum complexity: Collision and element distinctness with small range. *Theory of Computing*, 1(1):37–46, 2005.

[AW01] Andris Ambainis and Ronald de Wolf. Average-case quantum query complexity. *Journal of Physics A: Mathematical and General*, 34(35):6741, 2001.

[BDCG89] Shai Ben-David, Benny Chor, and Oded Goldreich. On the theory of average case complexity. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 204–216. ACM, 1989.

[BN95] Yosi Benasher and Ilan Newman. Decision trees with boolean threshold queries. *Journal of Computer and System Sciences*, 51(3):495–502, 1995.

[BT06] Andrej Bogdanov and Luca Trevisan. Average-case complexity. *CoRR*, abs/cs/0606037, 2006.

[BW02] Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002.

[DFO10] Anand K Dhulipala, Christina Fragouli, and Alon Orlitsky. Silence-based communication. *Information Theory, IEEE Transactions on*, 56(1):350–366, 2010.

[DJO+12] Hirakendu Das, Ashkan Jafarpour, Alon Orlitsky, Shengjun Pan, and Ananda Theertha Suresh. On the query computation and verification of functions. In *Proceedings of the 2012 IEEE International Symposium on Information Theory (ISIT)*, pages 2711–2715. IEEE, 2012.

[KK10] Hemant Kowshik and PR Kumar. Optimal ordering of transmissions for computing boolean threshold functions. In *Proceedings of the 2010 IEEE International Symposium on Information Theory (ISIT)*, pages 1863–1867. IEEE, 2010.

[NW99] Ashwin Nayak and Felix Wu. The quantum query complexity of approximating the median and related statistics. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 384–393. ACM, 1999.

[San95] Miklos Santha. On the monte carlo boolean decision tree complexity of read-once formulae. *Random Structures & Algorithms*, 6(1):75–87, 1995.

[Wan97] Jie Wang. Average-case computational complexity theory. *Complexity Theory Retrospective II*, pages 295–328, 1997.

[Weg87a] Ingo Wegener. *The complexity of Boolean functions*. Wiley-Teubner, 1987.

[Weg87b] Ingo Wegener. The complexity of symmetric boolean functions. In *Computation Theory and Logic, In Memory of Dieter Rödding*, pages 433–442, 1987.